AFRL-HE-WP-TR-2004-0110

United States Air Force Research Laboratory

Air Force Genomics, Proteomics, Bioinformatics System

DataCap-Data Collection Module

Phase 1-Development

Christopher Geib

GEO-CENTERS, INC.
P.O. BOX 31009
DAYTON, OH 45437-0009

John M. Frazier
Robert S. Cook

HUMAN EFFECTIVENESS DIRECTORATE
BIOSCIENCES AND PROTECTION DIVISION
APPLIED TOXICOLOGY BRANCH
WRIGHT-PATTERSON AFB, OH 45433-7400

July 2004

Final Report for November 2003 – March 2004

Human Effectiveness Directorate
Biosciences and Protection Division
Applied Toxicology Branch
Wright-Patterson AFB, OH 45433-7400

NOTICES

When US Government drawings, specifications or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Air Force Research Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Service
8725 John J. Kingman Rd., Ste 0944
Ft. Belvoir, Virginia 22060-6218

DISCLAIMER

This Technical Report is published as received and has not
been edited by the Technical Editing Staff of the Air Force Research Laboratory.

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-2004-0110

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE DIRECTOR

//SIGNED//

MARK M. HOFFMAN
Deputy Chief, Biosciences and Protection Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
July 2004	Final Tech Report	November 2003-March 2004

4. TITLE AND SUBTITLE

Air Force Genomics, Proteomics, Bioinformatics System
DataCap-Data Collection Module
Phase 1-Development

5a. CONTRACT NUMBER
F33615-00-C-6060

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62202F

6. AUTHOR(S)

*Geib, Christopher, **Frazier, John M.;**Cook, Robert S.

5d. PROJECT NUMBER
1710

5e. TASK NUMBER
1710D

5f. WORK UNIT NUMBER
1710D425

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

*Geo-Centers, Inc.
P.O. Box 31009
Dayton, OH 45437

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

**Air Force Research Laboratory
Human Effectiveness Directorate
Biosciences and Protection Division
Wright-Patterson AFB, OH 45433

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/HEPB

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-HE-WP-TR-2004-0110

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited..

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Cell biology studies involving genomics, proteomics and metabolomics technologies generate large quantities of data. Unfortunately, today's research environment is encumbered by inefficient access to this vital data. Information searches are hampered by disparities on both technical and conceptual levels between individual data sources. The current trend is for data sources to exist as a series of isolated computational silos, providing a depth of data in a narrow field of research. The Acero Genomics Knowledge Platform (GKP) is an enterprise solution that offers fully integrated data-representation across diverse scientific data types and sources. The DataCap data collection module (a part of the Air Force Genomics, Proteomics, Bioinformatics System (AFGPB) application is one module of a series of modules that operate on top of the Acero Platform. The purpose of the DataCap is to provide the individual researcher with the ability to collect experimental data in a integrated format compatible with the Acero GKP. This technical report covers the architecture, the design and the operation of the DataCap in its Phase 1 configuration.

15. SUBJECT TERMS

Genomics Proteomics Bioinformatics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John Schlager
Unclassified	U	U	SAR	36	19b. TELEPHONE NUMBER *(Include area code)* 937-904-9536

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK.

ABSTRACT

Cell biology studies involving genomics, proteomics, and metabolomics technologies generate large quantities of data. Unfortunately, today's research environment is encumbered by inefficient access to this vital data. Information searches are hampered by disparities on both technical and conceptual levels between individual data sources. The current trend is for data sources to exist as a series of isolated computational silos, providing a depth of data in a narrow field of research. The Acero Genomics Knowledge Platform (GKP) is an enterprise solution that offers fully integrated data-representation across diverse scientific data types and sources. The DataCap data collection module (a part of the Air Force Genomics, Proteomics, Bioinformatics System (AFGPB)) application is one module of a series of modules that operate on top of the Acero platform. The purpose of the DataCap is to provide the individual researcher with the ability to collect experimental data in an integrated format compatible with the Acero GKP. This technical report covers the architecture, the design and the operation of the DataCap in its Phase I configuration.

TABLE OF CONTENTS

Air Force Genomics, Proteomics, Bioinformatics System Phase I Development Effort ..... *i*

NOTICE ii

ABSTRACT iii

LIST OF FIGURES AND TABLES v

PREFACE vi

SUMMARY 1

1.0 INTRODUCTION *7*

1.1 Purpose 7

1.2 Scope 7

2.0 METHODS, ASSUMPTIONS, AND PROCEDURES 8

2.1 Software and Equipment 8

2.2 Programming Methodology 9

3.0 DISCUSSION 10

3.1 The Bioinformatics Data Problem 10

3.2 The Bioinformatics Data Solution 10

3.3 Selection of the Acero Genomics Knowledge Platform 11

3.4 AFGPB DataCap Primary Classes 12

3.5 AFGPB DataCap GKP Model Extension 14

3.6 AFGPB DataCap Module Program Flow 15

3.6.1 Activities Tab 19

3.6.2 Experiments Tab 23

3.6.3 e-Lab Notebook Tab 29

4.0 CONCLUSIONS 34

5.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS 35

LIST OF FIGURES AND TABLES

Figure 1.	Silos of Data	7
Figure 2.	Acero n-Tiered Server Architecture	8
Figure 3.	The AFGPB DataCap Module Primary Classes	12
Figure 4.	Experiment as an Extension of the BOM	14
Figure 5.	AFGPB DataCap Module Program Flow – Initial Start Up	16
Figure 6.	The Acero DDK on Startup	17
Figure 7.	AFGPB DataCap Module Initial Screen Display	18
Figure 8.	AFGPB DataCap Module Main Panel Class Program Flow	19
Figure 9.	AFGPB DataCap Module Activities Tab Display	20
Figure 10.	AFGPB DataCap Module Project Panel Class Program Flow	21
Figure 11.	AFGPB DataCap Module Experiments Tab Display	24
Figure 12.	AFGPB DataCap Module Experiment Panel Class Program Flow	25
Figure 13.	AFGPB DataCap Module Experiment Panel Class Program Flow Continued	27
Figure 14.	AFGPB DataCap Module e-Lab Notebook Tab Display	29
Figure 15a.	AFGPB DataCap Module e-Lab Book Panel Class Program Flow	30
Figure 15b.	AFGPB DataCap Module e-Lab Book Panel Class Program Flow Continued	32

PREFACE

This non-peer-reviewed report describes the architecture, design, and operation of the DataCap data collection module, the first of several application modules that are a part of the Air Force Genomics, Proteomics, Bioinformatics System to be integrated with the Acero Genomics Knowledge Platform. This report covers the Phase I effort that started November 2003, and completed March 2004 under Department of the Air Force Contract number F33615-00-6060. Dr. John Schlager served as the Contract Technical Monitor for the U.S. Air Force, Air Force Research Laboratory, Cellular Dynamics and Engineering, AFRL/HEPB, and Dr. Darol Dodd served as Program Manager for the ManTech/Geo-Centers Joint Venture. All work was performed at Wright-Patterson AFB, OH, in the Applied Biotechnology Branch, Protection and Bioscience Division, Human Effectiveness Directorate of the Air Force Research Laboratory.

Acknowledgement of Sponsorship

Effort sponsored in whole or in part by the Air Force Research Laboratory, USAF, under Memorandum of Understanding/Partnership Intermediary Agreement No. FA8652-03-3-0005. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

Disclaimer

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the Air Force Research Laboratory.

1.0 INTRODUCTION

1.1 Purpose

This report provides details concerning the background, design and operation of the Air Force Genomics, Proteomics, and Bioinformatics System (AFGPB) DataCap data collection module Phase I development effort.

1.2 Scope

The scientific studies of genomics and proteomics are highly complex endeavors, requiring the acquisition of large quantities of diverse data. The sources of data accessed by the Applied Biotechnology Branch include laboratory notebooks, computer driven instrumentation, publication databases, as well as online peer reviewed articles. Each of these sources creates for themselves, a "silo" of data. Data in each silo is stored differently, whether electronically or on paper, and generally in a format that is uniquely designed for that given data source (Figure 1). As such, it is these very silos that create conditions of incompatibility of data sources, which in turn create barriers to access, and ultimately, obstacles to collaboration. All of these issues compound the ability of researchers to conduct meaningful scientific research and arrive at useful conclusions.

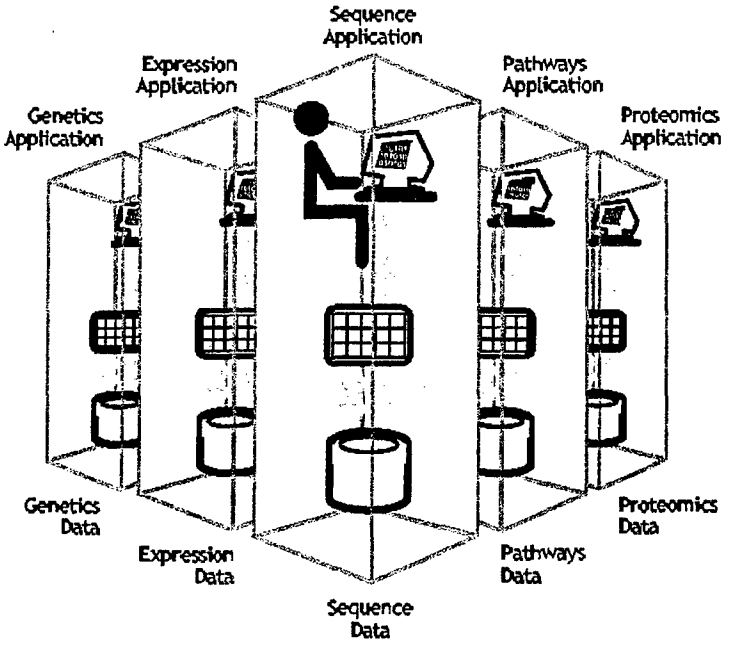

Figure 1. Silos of Data

To solve this problem, a means was required to capture data from all these different sources into a single integrated "virtual" datastore. The solution was the selection of the Acero Genomics Knowledge platform (GKP) as the way to integrate all of the disparate data sources. The Acero framework provides both a broad standard for biological data, and a flexible and streamlined approach to software development.

2.0 METHODS, ASSUMPTIONS, AND PROCEDURES

2.1 Software and Equipment

The Acero platform, upon which the Air Force Genomics, Proteomics, and Bioinformatics System (AFGPB) is built, is itself built using advanced, model-driven application server technology to create a modular, scalable, and fault-tolerant server with multi-tier architecture. The server functionality is contained within the isDiscoveryCenter which in turn contains a number of modular components (Figure 2):

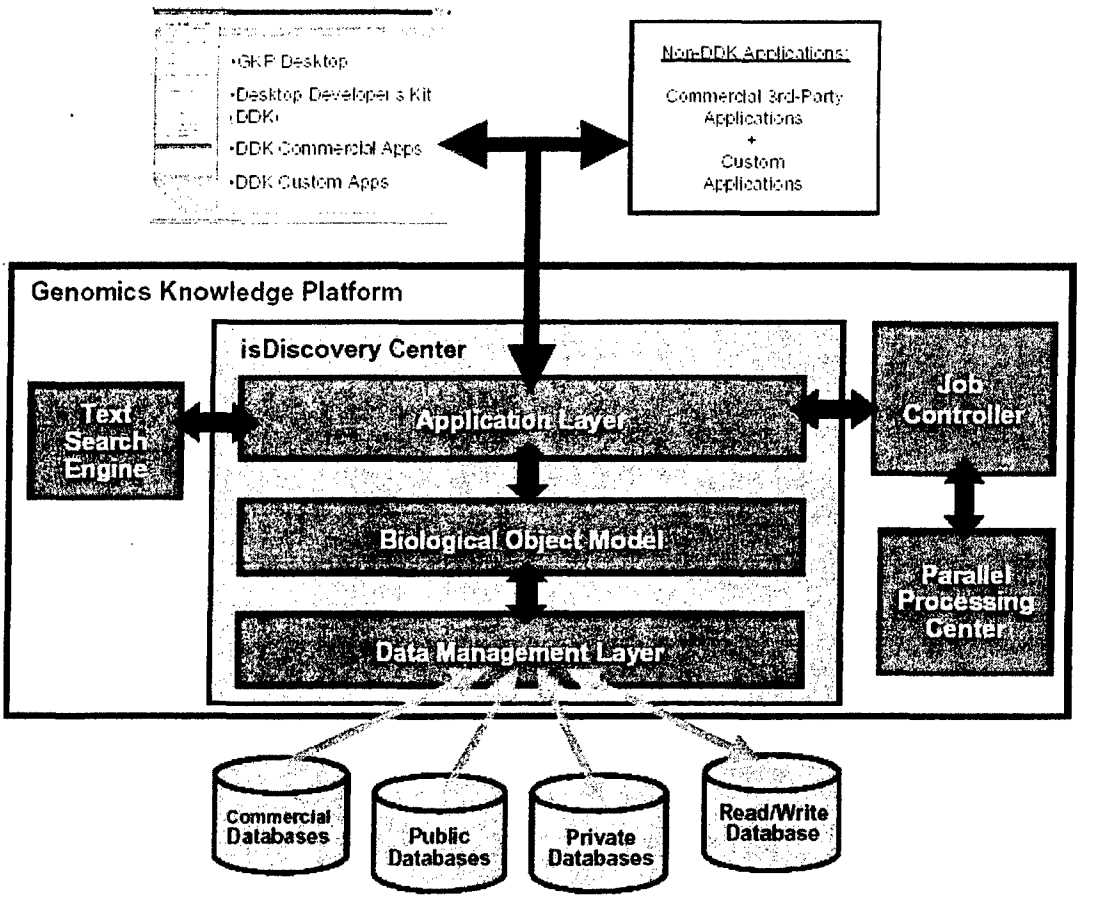

Figure 2. Acero n-Tiered Server Architecture

The GKP and the server module applications reside presently on one desktop PC (development system) and on the HEPB Bioinformatics Data server (production system). The desktop PC is an IBM clone using a Pentium 4 processor at 3GHz and having 512MB of random access memory.

The Bioinformatics Data server is a Dell 4600 server with two 2GHz Xeon processors and 2GB of random access memory. In addition, the server has an attached RAID 5 array with a useable storage capacity of 1.64 terabyte (TB). The Dell 4600 server is expandable to four Xeon processors and 8GB of memory, and currently there are expansion slots and ports to add up to 4 more RAID 5 storage arrays.

The server core component, called the Secant Extreme Distributed Service Coordinator (DSC) is a proprietary C++ based set of modules. The Genomics Knowledge Platform (GKP), as well as all of the DDK custom applications (like DataCap) are Java language based. The DDK provides a rich set of graphical user interface (GUI) tools and a common entrance point to the isDiscoveryCenter Application Programming Interface (API).

The development tools that are being used to develop the DataCap and the other components of the Laboratory Data Integration and Management System include the Rational Rose Uniform Modeling Language (UML) application, the Oracle JDeveloper Java development application, and the Acero Secant Object Data Language Compiler (ODLC). The Rational Rose UML application is required to model extensions to the GKP schema. The UML is used in conjunction with the ODLC which converts the UML output into SQL script for creating tables within the database, as well as Java code files that create the object class representing the datastore to include both Get and Set methods for data flow. The ODLC operates via an Acero provided plugin to the Rational Rose application. JDeveloper provides the means to create GUI applications employing the Java libraries as well as the Acero DDK libraries. The DataCap module and all subsequent modules will be built using JDeveloper. JDeveloper was chosen because it is a component of the Oracle Application Tools set that is site-licensed by the USAF at Wright-Patterson AFB. The ODLC is an Acero provided tool that has the built-in logic to correctly define the required fields in a new table, as well as generate the appropriate set of Java methods for setting data into the new table and retrieving data from the new table.

The DataCap is a module that runs within the Acero Desktop Developers Kit (DDK) (Figure 2) as a DDK custom application. DataCap is written in Java version 2, release 1.4.2. A complete set of the source code files are provided in the Appendix.

Underlying all of the applications, APIs and tools is the Oracle database. The database employed is the Oracle 9i Release 2 version and is installed on both the development PC as well as the Bioinformatics Data server.

2.2 Programming Methodology

The programming methodology for DataCap is the use of Java as the programming language. Java is accepted by the Department of Defense, has excellent security, and is highly portable to a large array of platforms and operating systems. DataCap started out as an application called GPB, developed by Procter & Gamble. The GPB application was intended as a corporate "do-all" application and fell short of the needs of researchers at AFRL. Additionally, much of the application was Java Beans based and it was desired to have an application that was more client-server oriented. As such, DataCap captured a number of positive features, e.g., the file system provisions built into GPB, but converted the Java code to operate on a client-server basis. The DataCap code was further modified to add data collection aspects such as the e-Lab notebook

feature. Additionally, the original GPB did not capture a number of unique features found in the AFRL biotechnology laboratories. These features required extension of the Biological Object Model of the GKP by creating a new table and Java class called Experiment. The successful implementation of the Experiment table and Java class into the DataCap Phase I effort opens the way to easily extend the model further during DataCap Phase II development. In the following Discussion section, details of the architecture and design of the DataCap module will be described.

3.0 DISCUSSION

3.1 The Bioinformatics Data Problem

Within the Applied Biotechnology Branch of the Human Effectiveness Directorate, there exists a problem that is common throughout the biotechnology research industry, that being the incompatibility and disparate sources of biological data. This problem creates a situation wherein there are barriers to access data, and consequently, obstacles to collaboration. Examples of the disparities of data include such things as individual lab notebooks, computer-aided instrumentation with their own unique ways of storing and managing data results, and literature searches that result in paper as well as electronic file type documents. None of this data can or is stored together in any ready to use or analyze way. Because much of it is stored offline in various locations, attempts at collaborating in near real time within the lab or with external partners is impossible. In addition to the incompatibilities and the disparate locations for data, bioinformatics data takes up ever increasing storage space. Clearly, a need exists to bring all of this data together in such away as to make it readily available not only internally, but available to the various collaborators and partners involved in research for the Applied Biotechnology Branch.

3.2 The Bioinformatics Data Solution

The solution is to build an integrated data collection, storage and management system whereby all research data, regardless of source, is located in one central repository. To accomplish this requires not just a network user friendly interface, but a means to automatically capture as much of the data as possible, parse it to a specific, reliable, and approved format, and store it in the context of research activity and experiment. User interfaces must exist to enter manually acquired data such as that found in laboratory notebooks. In addition to the user interface and the tools to acquire the remote data, a robust, secure, and reliable database engine or engines are required.

The original design strategy was to create a system design that involved five distinct steps. These steps were:

- Design and create a complete data model based on the science being performed within the Applied Biotechnology branch. The approved data model would become the relational database schema.

- Create or acquire an Application Programming Interface (API) to provide the platform for the user interfaces and various programs that were to be developed

for the integrated solution.

- Develop a Data Collection module that would capture data from all the disparate sources such as laboratory notebooks, computer driven lab instrumentation (e.g., spectrophotometers and chromatographs), and literature sources. The Data Collection module must meet the needs of the individual researcher by addressing individual activities and experiments associated with these activities.

- Develop a Data Manipulation module that performs queries on the collected data, is capable of doing data mining, and creates data arrays for submission to external data processing applications such as MatLab, SAS, or modeling software on supercomputing assets.

- Create a Data Management module that provides overall integrated management and control for all bioinformatics research within the Applied Biotechnology branch. The Data Management module is designed to serve the multiple layers of the research organization.

3.3 Selection of the Acero Genomics Knowledge Platform

Initial research and literature search in July 2003 for possible Commercial-Off-The-Shelf (COTS) solutions for an integrated bioinformatics management solution led to the discovery of the Acero Genomics Knowledge Platform (GKP). This platform was in its early stages of use within the Genomics Research Infrastructure Partnership (GRIP), a partnership including the Genomics Research Institute of the University of Cincinnati, Children's Hospital, Wright State University, Procter & Gamble, and Acero. After some additional research, it was decided that the Applied Biotechnology Branch would likewise employ the Acero platform.

One primary benefit to employing the Acero GKP was the need to design and create the data model and relational database schema had already been accomplished. The Acero platform utilizes an underlining in-silico definition of biological processes through the Biological Object Model (BOM). The BOM describes traditional sequence analysis, large-scale expression analysis, biological sample description, molecular pathways, polymorphism data, and several other related areas of life science research. Additionally, the Acero BOM is readily extendable, thus allowing for the creation of a unique design that is consistent with the environment found within the local laboratory environment.

Another benefit of the Acero platform is the application programming interface was already created and validated. The API enables an application developer to write generic applications using the Java programming language that do not rely on a single source of data and at the same time make the integrated data available to a broad range of COTS data analysis tools. The Acero platform includes the Desktop Developers Kit (DDK), which is a user interface to the underlying model and API.

As a result of selecting the Acero platform, five distinct development steps were reduced to three. The Data Collection module, the Data Manipulation module and the Data Management

module are the three development activities required. The balance of this report addresses the first Phase of the Data Collection module, also called DataCap.

3.4 AFGPB DataCap Primary Classes

The DataCap Phase I application software design is based in part on the original work done by Procter & Gamble in their GPB application for the GRIP. However, as previously stated, DataCap takes a number of supporting Java classes out of the Java Beans category and moves them into the client-server context. At present there are a total of 29 Java files (or Java classes) associated with DataCap. The primary classes which drive the application are shown in Figure 3.

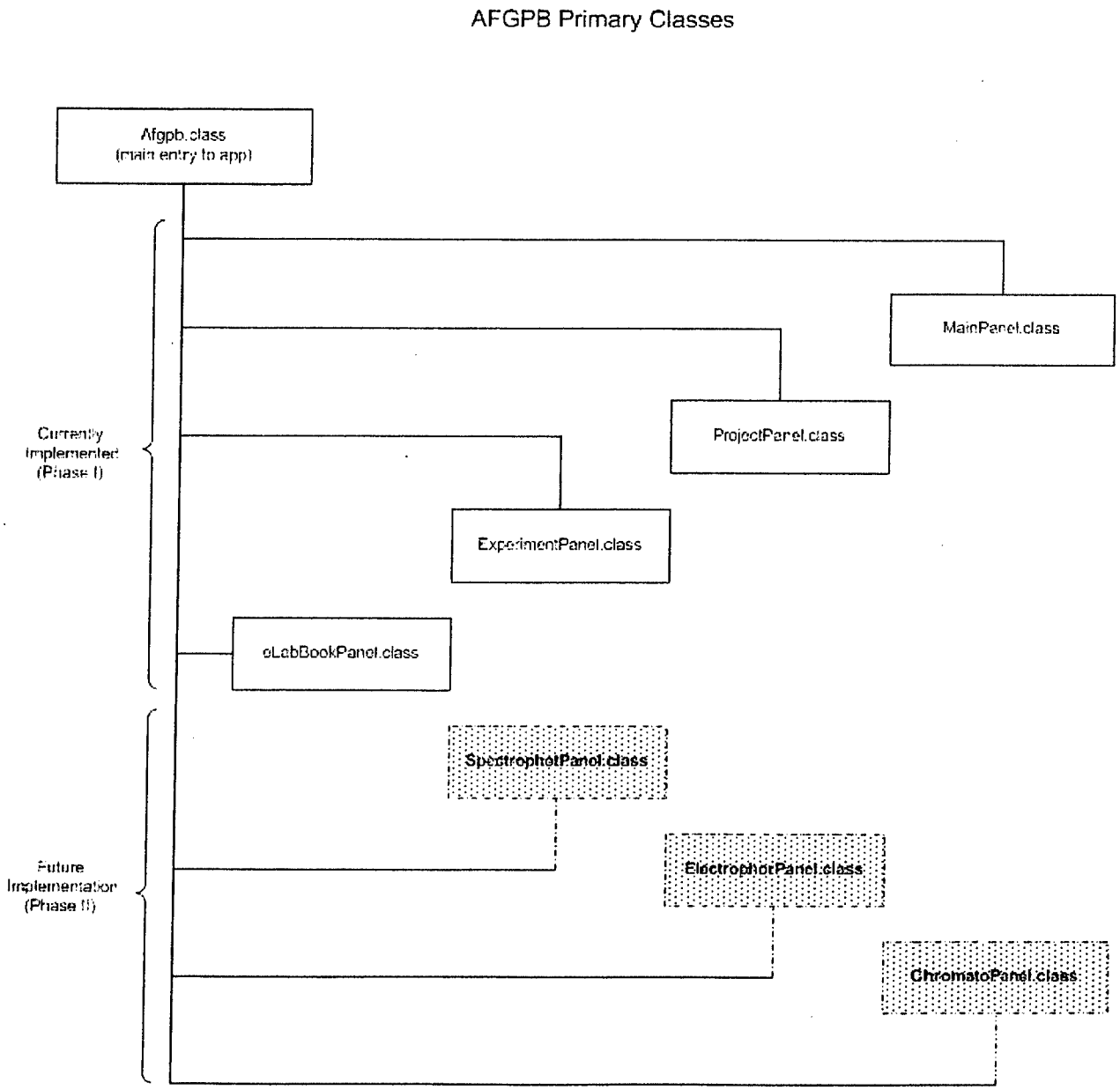

Figure 3. The AFGPB DataCap Module Primary Classes

As can be seen, the main entry into the program is via the Afgpb.class (the compiled Java name). Within Afgpb.class, a basic multi-tabbed Pane is created. This pane uses the DDKJTabbedPane which is one of the numerous GUI tools built into the DDK and is directly derived from the Java JTabbedPane class. The Afgpb.class also instantiates a Java Data Objects (JDO) Persistence Manager, as well as a JDO Transaction Manager. Both of these objects are used throughout the various object classes in the application to connect and transfer data via the API to the datastore (database). The DDKJTabbedPane hosts each of the panels attached to each tabbed pane. Within Afgpb.class is a switch-case method that triggers the appropriate panel and class object for a specific feature.

In Figure 3, the primary classes are shown as well as the future implementation classes. Afgpb.class, MainPanel.class, ProjectPanel.class, ExperimentPanel.class, and eLabBookPanel.class comprise the main Phase I Java classes, while SpectrophotPanel.class, ElectrophotPanel.class, and ChromatoPanel.class will be implemented in Phase II. The functions of the primary Phase I classes are as follows:

- MainPanel.class – Provides the presentation of the equivalent of a "home" screen with an imbedded image and a button. The button, an instance of a utility class called SupportButton, starts Adobe Acrobat Reader on the client machine and opens a file that is the current support manual and is located on the server.

- ProjectPanel.class – Displayed as the Activities tab to the user, the ProjectPanel class presents the means to create new research Activities with a description, a Work Unit number, an activity leader and authorized activity members. The Activities Save button fires Java methods that create a folder in the GKP File System (GKPFS).

- ExperimentPanel.class – The ExperimentPanel class presents the user interface to create and edit Experiments associated with a given Activity. One of the main features of this class is its interface with the extended Experiment.class and the Experiment table within the Oracle database. As Experiments are created, an entry is made through Experiment.class Set methods to add or update data to fields. Additionally, the ExperimentPanel class further interfaces with the GKPFS to create additional subfolders and sub-subfolders under the appropriate Activity. The ExperimentPanel class represents a significant modification to the original GPB application.

- eLabBookPanel.class – This class is a complete departure from the original GPB developed by P&G Pharmaceuticals. This source of important research data is integrated into the DataCap, and provides the portal to create a text-only entry as found in a typical lab notebook. Due to current limitations in Java, the ability to paste a graphical image is not available. The underlying data structure being used is based on a VARCHAR2 field in the Annotation table. Additional methods provide the means to display the contents of all lab notebooks specific to the displayed activity in an integrated fashion. A method is provided to save the contents of an integrated lab notebook as a Microsoft Word document.

3.5 AFGPB DataCap GKP Model Extension

In the design of the Biological Object Model (BOM) within the GKP, most of the science related classes inherit from class Publishable. The BOM is designed to be extensible through additions of classes that are children of the Publishable class. The existing BOM does not adequately cover the concept of an experiment as understood within the Applied Biotechnology Branch. For this reason, the BOM has been extended. Figure 4 details this extension.

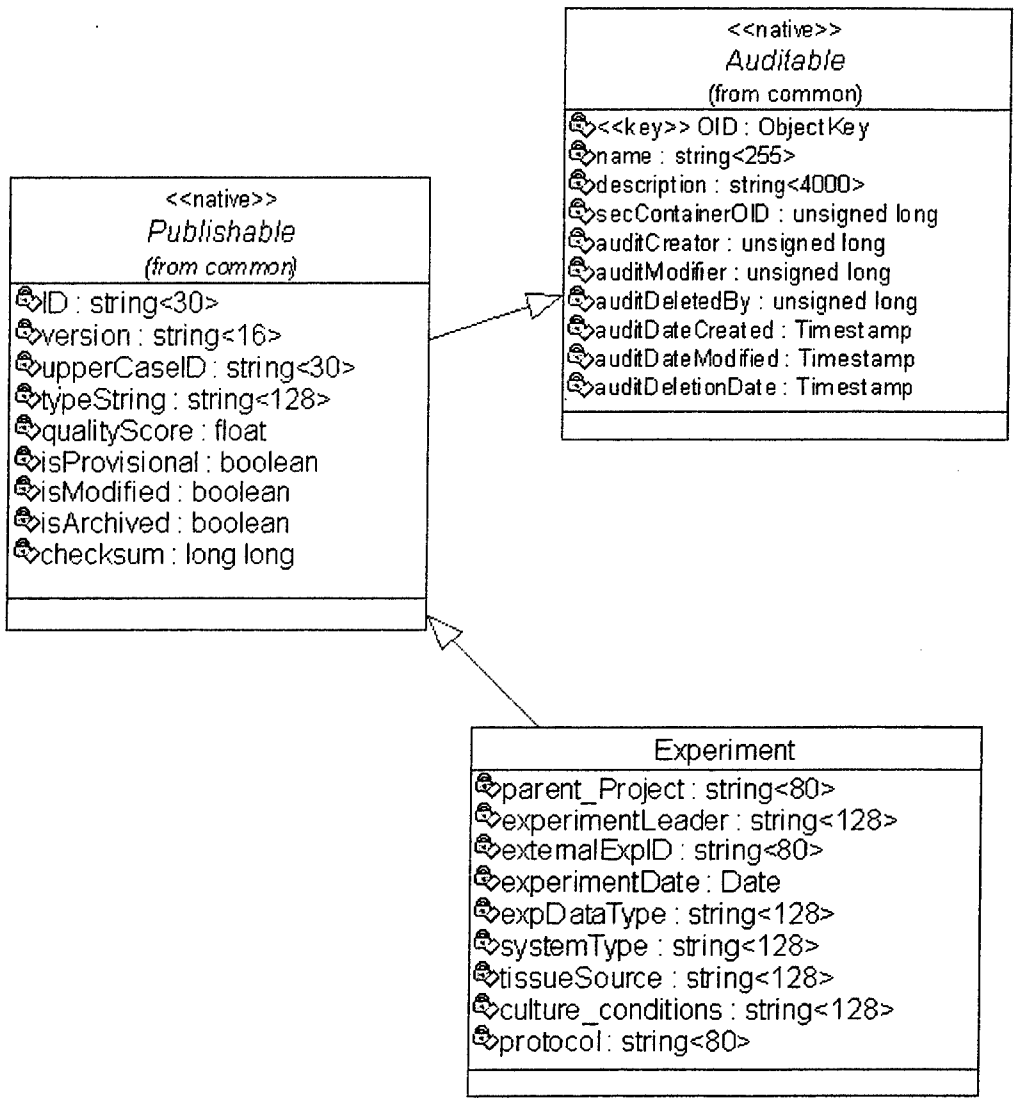

Figure 4. Experiment as an Extension of the BOM

To extend the model requires the use of the Rational Rose Uniform Modeling Language tool, along with the Acero Secant Rose plug-in. To properly extend the Acero model, any class object must be a child of Publishable, which in turn is a child of Auditable. Additionally, the translation algorithm in the ServerBean.class calls methods that are a part of the Hit class as well. This requires the addition of a number of rows that are not shown in the above graphic. Although

required by the system, they are not germane to data collection and manipulation. For a complete listing of the fields in the Experiment table, see the listing in the Appendix. The model diagram above details the classes within the BOM. As such, there are tables that correspond to the classes within the GKP except virtual classes. Virtual classes do not have corresponding tables. The Acero plug-in when run in Rational Rose generates a number of files. These files include Java class files ready for compiling, a Metadata Object File, or MOF file, and a schema file. The Java files are compiled along with all the other class files in the DataCap implementation. The MOF when read in by the GKP, is used to register the extended class and defines the class to the system. Under the current build of the GKP, two Java files are created and later registered to the GKP. One defines the extended object in Java Data Object (JDO) format, and the other defines the extended object as a Java Bean Object (JBO). The schema file is in a ready format to be run on the Secant schema generator application that converts the schema file to a SQL file and then executes PL*SQL to run the script to create a table within the database. Within DataCap, the Experiment object represents an actual table within the Oracle database.

3.6 AFGPB DataCap Module Program Flow

The next paragraphs will discuss in detail, using both figures and text, the overall flow and logic of the DataCap. Figure 5 details the initial startup of DataCap.

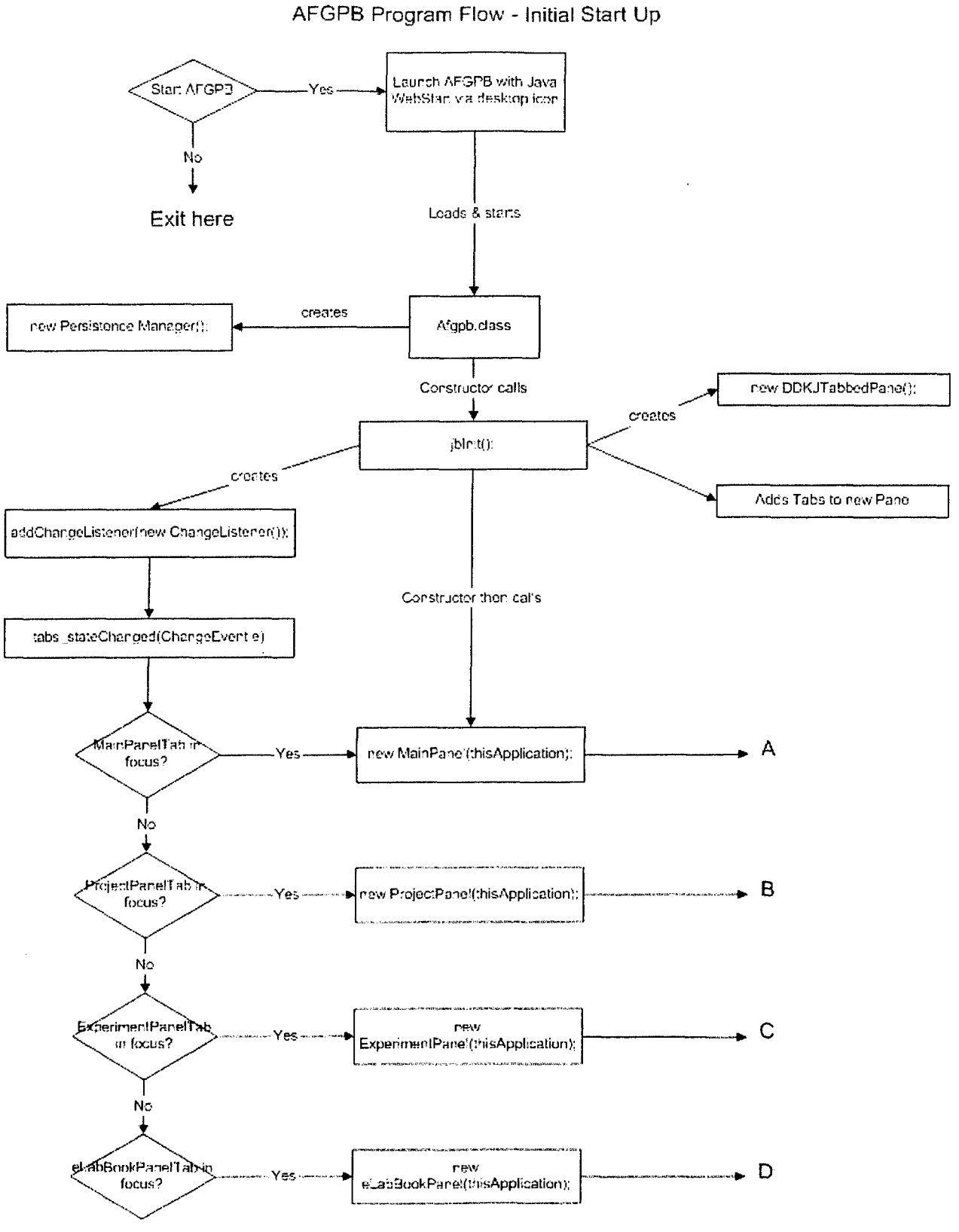

Figure 5. AFGPB DataCap Module Program Flow – Initial Start Up

Figure 5 assumes that the user computer has had the latest Java Web Start (version 1.4.2) installed, and that there is a GKP icon on the user computer Desktop. Additionally, it is assumed that the user has an account and a password on the GKP system. When the user first double clicks the GKP application icon on the desktop, an indirect application launches Java Web Start. Java Web Start in turn loads and starts the Java Jar executable archive file that creates the DDK on the desktop. The DDK is the foundation application in which all GKP applications are run. The DDK appears on the desktop as in Figure 6.

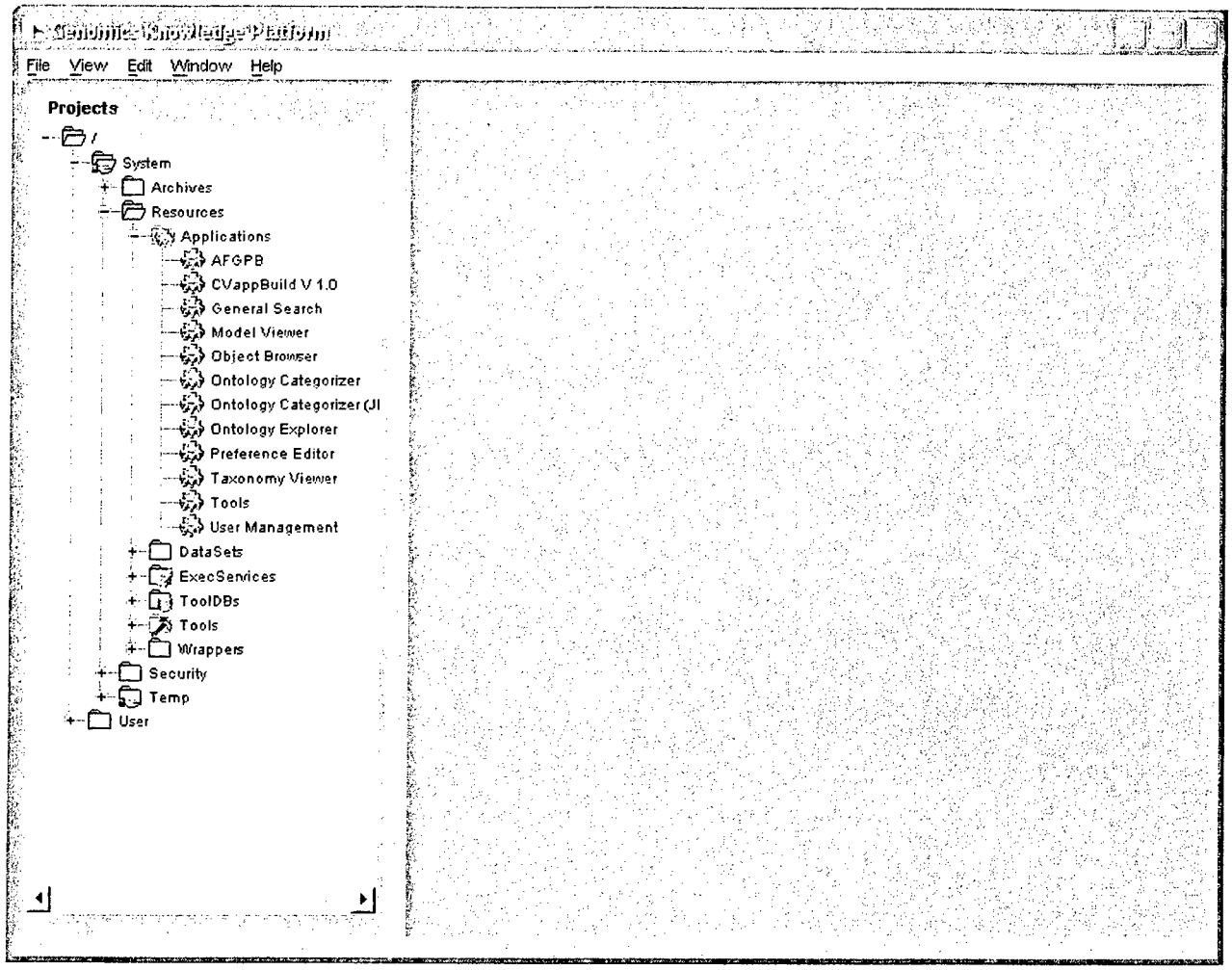

Figure 6. The Acero DDK on Startup

Note in the left hand panel labeled Projects in Figure 6, is a hierarchical tree with the Resources Applications area expanded. The first icon listed is AFGPB. When this icon is double clicked, the DDK loads and executes the file **afgpb.jar**. In addition to configuration files and resource files, the afgpb.class is loaded and executed. As afgpb is constructed, it creates in memory a Persistence Manager, which is required for communications and data flow between the Java application and the datastore which is an Oracle 9i database. The constructor also calls the **jbInit()** Java method which is created in code when using the Oracle JDeveloper tool for creating GUI Java applications. Within this Java method are all the necessary calls to create a GUI window and then create and place all the controls and text within the GUI window such as buttons, combo boxes, labels, text entry fields, etc. Additionally, the jbInit() method instantiates

several Java listeners, most notably the tab_stateChanged listener. When the constructor and its methods finish running, Figure 7 details what is visible to the user.

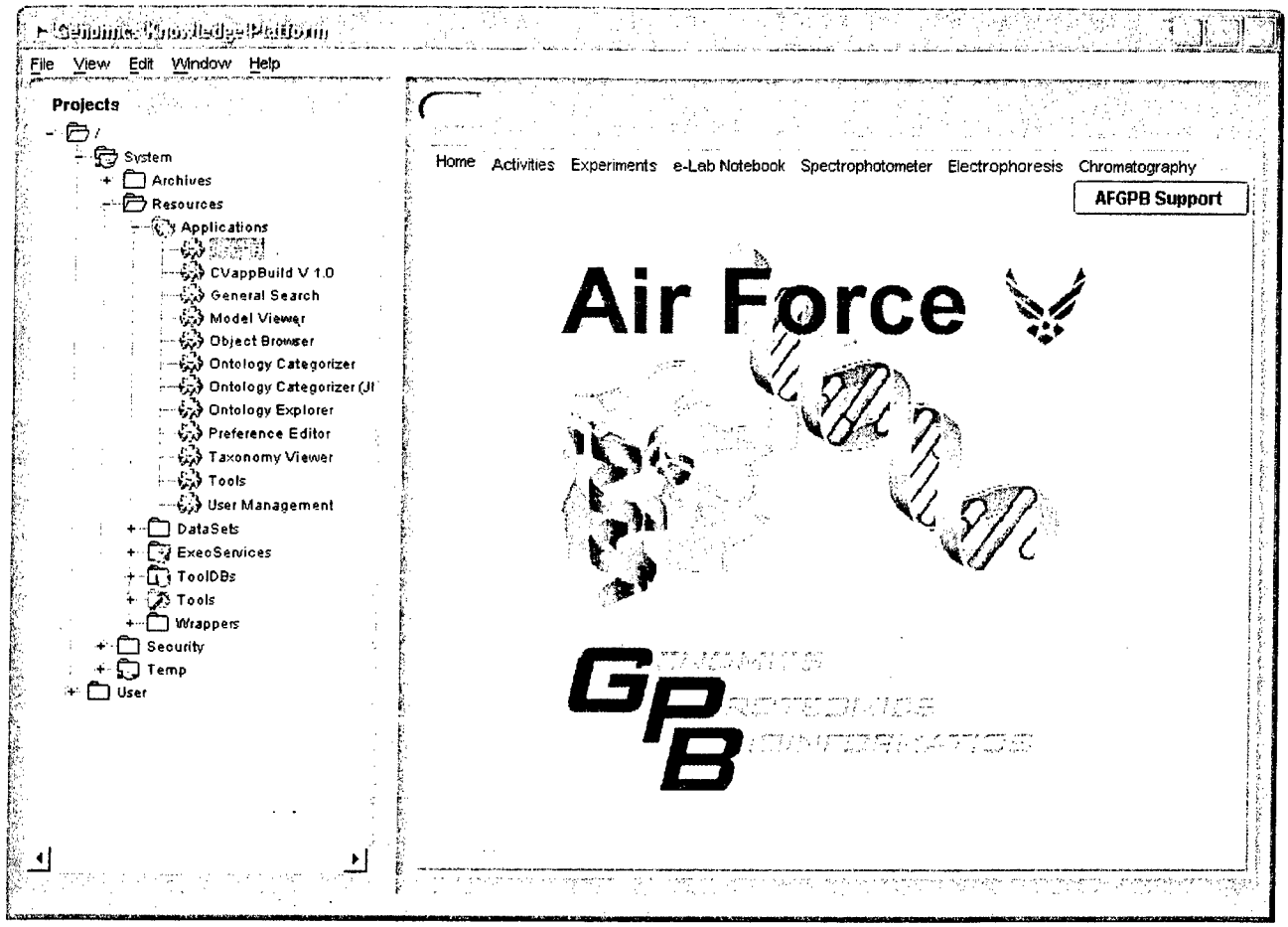

Figure 7. AFGPB DataCap Module Initial Screen Display

Looking at Figure 7, note the tabs along the top of the DataCap module. These tabs constitute hot spots for the tabs_stateChange listener. When a user clicks one of the tabs, the listener in turn fires a method that instantiates a new class. Looking back at Figure 5, the jbInit() method calls the MainPanel class by instantiating a new MainPanel object. Of the seven tabs currently displaying in AFGPB, only the first four: Home, Activities, Experiments, and eLab Notebook, when selected, call a subsequent Java object.

The program flow for the MainPanel class is detailed in Figure 8. The instantiation of a new instance of a MainPanel object starts the MainPanel.class. As with all Oracle JDeveloper built applications, MainPanel.class contains a jbInit() method call in the constructor of the class. In the MainPanel class, the method sets down a JPanel interface, creates and places a new SupportButton, and calls and places getImageIcon() that puts the graphic on the screen.

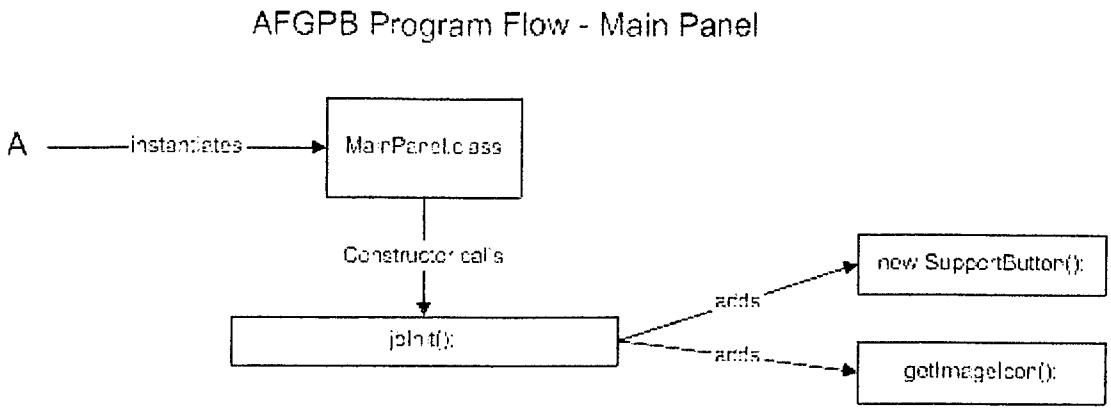

Figure 8. AFGPB DataCap Module Main Panel Class Program Flow

3.6.1 Activities Tab

When a user places the mouse cursor over the Activities tab and clicks the mouse, the ProjectPanelTab is brought into focus and the listener starts a new instance of the ProjectPanel class. The terminology of calling this portion of the flow a Project is carryover from the P&G version which referred to Activities (as found in the Applied Biotechnology Branch) as Projects. The user interface as found in Figure 9 presents itself visibly as Activities.

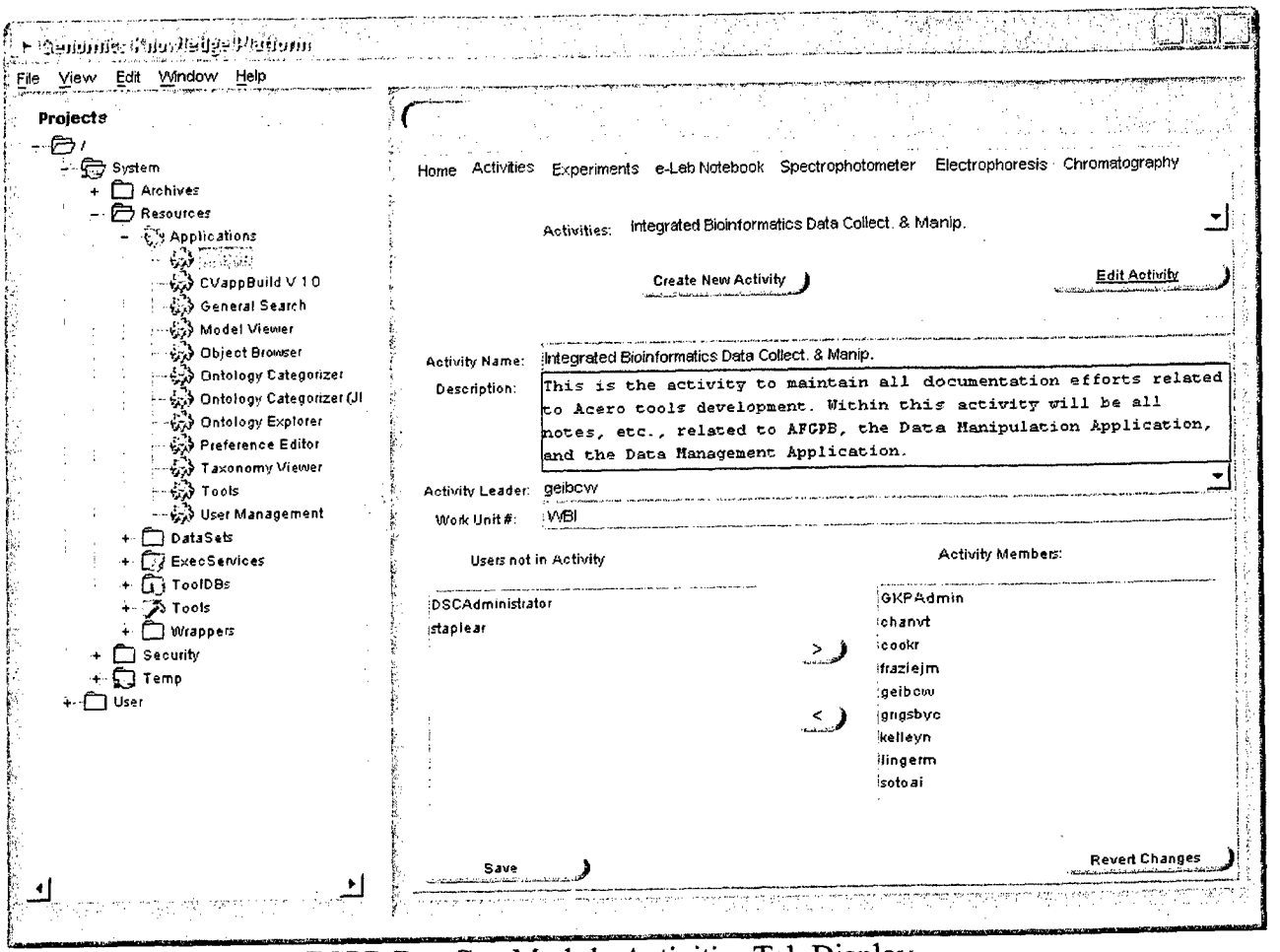

Figure 9. AFGPB DataCap Module Activities Tab Display

The program flow for the ProjectPanel class (displayed as the Activities Tab), is detailed in Figure 10.

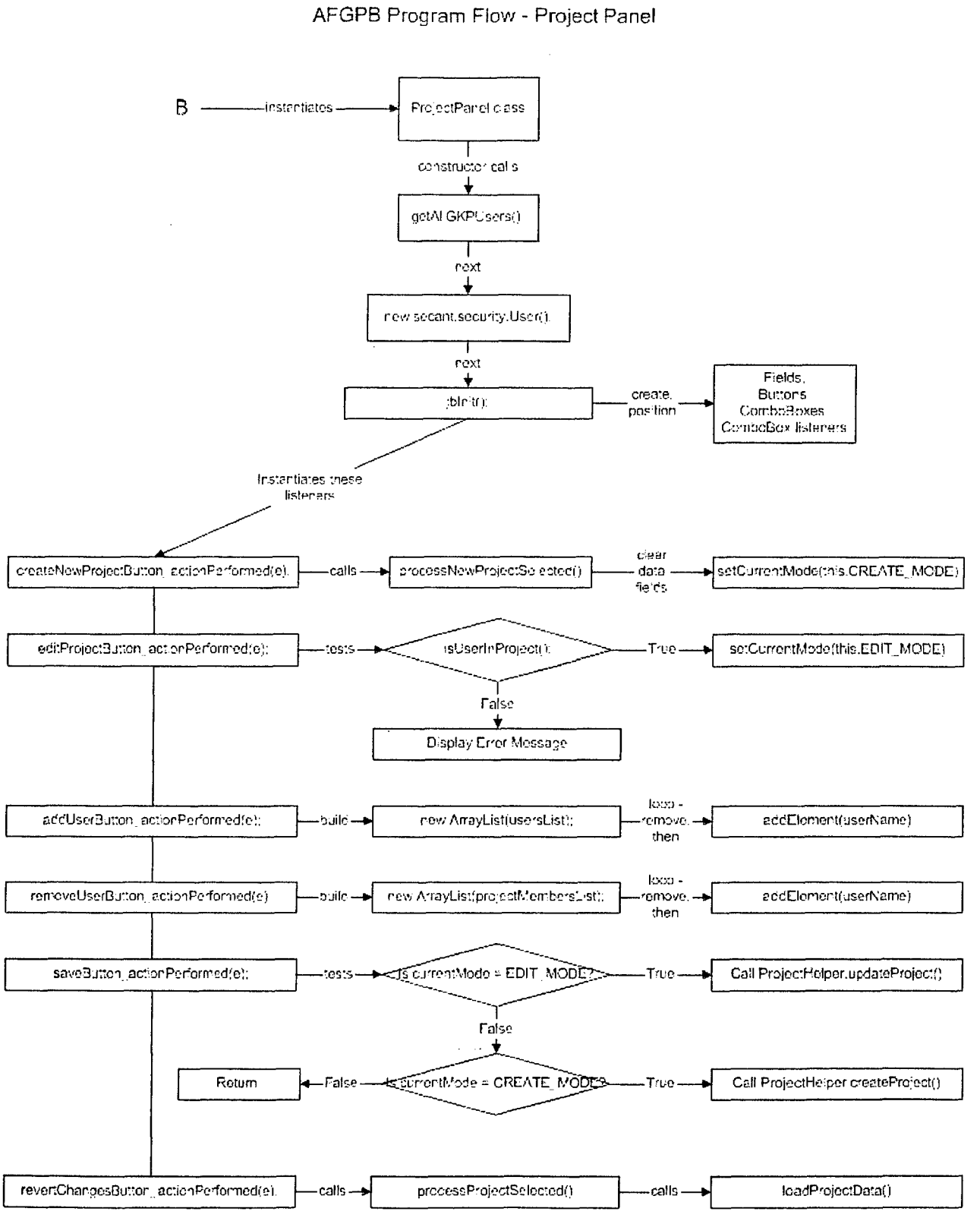

Figure 10. AFGPB DataCap Module Project Panel Class Program Flow

When the tabs_stateChanged listener in afgpb.class detects that a user has selected the Activities tab, the listener creates a call to a new instance of ProjectPanel. The ProjectPanel class first creates and then fills a Java collection by a call to getAllGKPUsers(). It then follows up by creating a new instance of a secant.security.User() object and assigning the result to a variable that is used within the class. Provided these actions are successful and do not throw any exceptions, the constructor then goes on to call the jbInit() method for the class. The jbInit method goes about creating a JPanel instance and then populates the panel with fields, buttons, combo boxes and instances of combo box listeners. These listener objects are created to detect changes in certain combo boxes and refresh other combo boxes and fields when a value changes. There are six buttons in the ProjectPanel class and the jbInit method creates a listener for each.

When the createNewProjectButton listener object detects that a mouse click was performed on the button, the listener calls the processNewProjectSelected() method. The processNewProjectSelected method in turn calls clearProjectData() which goes about clearing all the fields and combo boxes other than projectComboBox. The method re-calls the getAllGKPUsers method and repopulates the usersList DDKJList area. The clearProjectData method then returns a void to processNewProjectSelected, where the method calls setCurrentMode, passing the static integer constant CREATE_MODE. This causes all the fields that are editable to go from a grayed-out, non-editable state, to a fully editable state. In this state, the user (if authorized) can enter information about a new Activity to include the Activity name, a description up to 4000 characters, define the Activity work group leader, the AFRL Work Unit this activity bills to, and add a list of users to the authorized list of users for this activity.

The editProjectButton listener object detects that a mouse click has occurred on the Edit Activity button. This listener calls the isUserInProject method which attempts to determine if the logged in user that clicked the Edit Activity button is a member of the displayed activity. If the user is not a member of the displayed activity, then an error message is displayed to the user, and upon acknowledging the message, is returned to the Activity panel with no state changes. However, if the user is a member of the activity, then the listener changes the state of the panel by a call to setCurrentMode passing in the static integer constant EDIT_MODE. The state change allows the user to make changes to the Activity Name, Description, Work Unit number, and make changes to the membership of the Activity.

The addUserButton listener is linked to the button with the right facing arrow between the lists of user names. When this button is clicked by the mouse, the listener performs the following actions. It first makes a call to the ArrayList method and generates an array of selected (one or many) users from the usersList list box. Once this array is built, the listener then loops through the array and adds the individual user names via an addElement call to the projectMembersList list box.

The removeUserButton listener is linked to the button with the left facing arrow between the lists of user names. When this button is clicked by the mouse, the listener first makes

a call to the ArrayList method and generates an array of user names. The listener then processes the array via a loop and does a removeElement call as appropriate.

When the Save button is clicked by the mouse, the saveButton listener is called. The listener first tests to determine if the setCurrentMode constant is set to EDIT_MODE, and if true, the listener makes a call to the ProjectHelper.updateProject method. The ProjectHelper.updateProject method first does a check to determine if the root project directory exists in the DDK Workbench, and then updates the properties for the activity leader, activity organization, and activity description. On the other hand, if the setCurrentMode constant is not in EDIT_MODE, then the listener tests through an else if statement whether the setCurrentMode constant is set to CREATE_MODE. If the test results in a Boolean true, then the listener calls the ProjectHelper.createProject method. The ProjectHelper.createProject method, just like the upadateProject method, first determines if the root project directory exists in the DDK Workbench. The method then sets properties for the activity leader, organization, and description. If these method calls are all successful, then the listener adds the appropriate elements to the display, updates the Workbench screen so the new Activity folder appears, posts a dialog box notifying the user of success, and performs cleanup to ensure proper presentation of information to the user.

A mouse click on the Revert Changes button calls the revertChangesButton listener. The listener executes a call to the processProjectSelected method, which in turn calls the loadProjectData method. The loadProjectData method grabs the contents of the projectComboBox into a variable, and then sets the project name text field, description text area, the project leader combo box, and organization text field through a series of setText or setSelectedItem methods with appropriate getter methods passed in as arguments. The method then removes all the elements in the usersList list box as well as the projectMembersList list box. At this point, the method then proceeds to re-populate these list boxes with the information for the selected project.

3.6.2 Experiments Tab

Figure 11 below shows what the user sees when the Experiments tab is selected with the mouse.

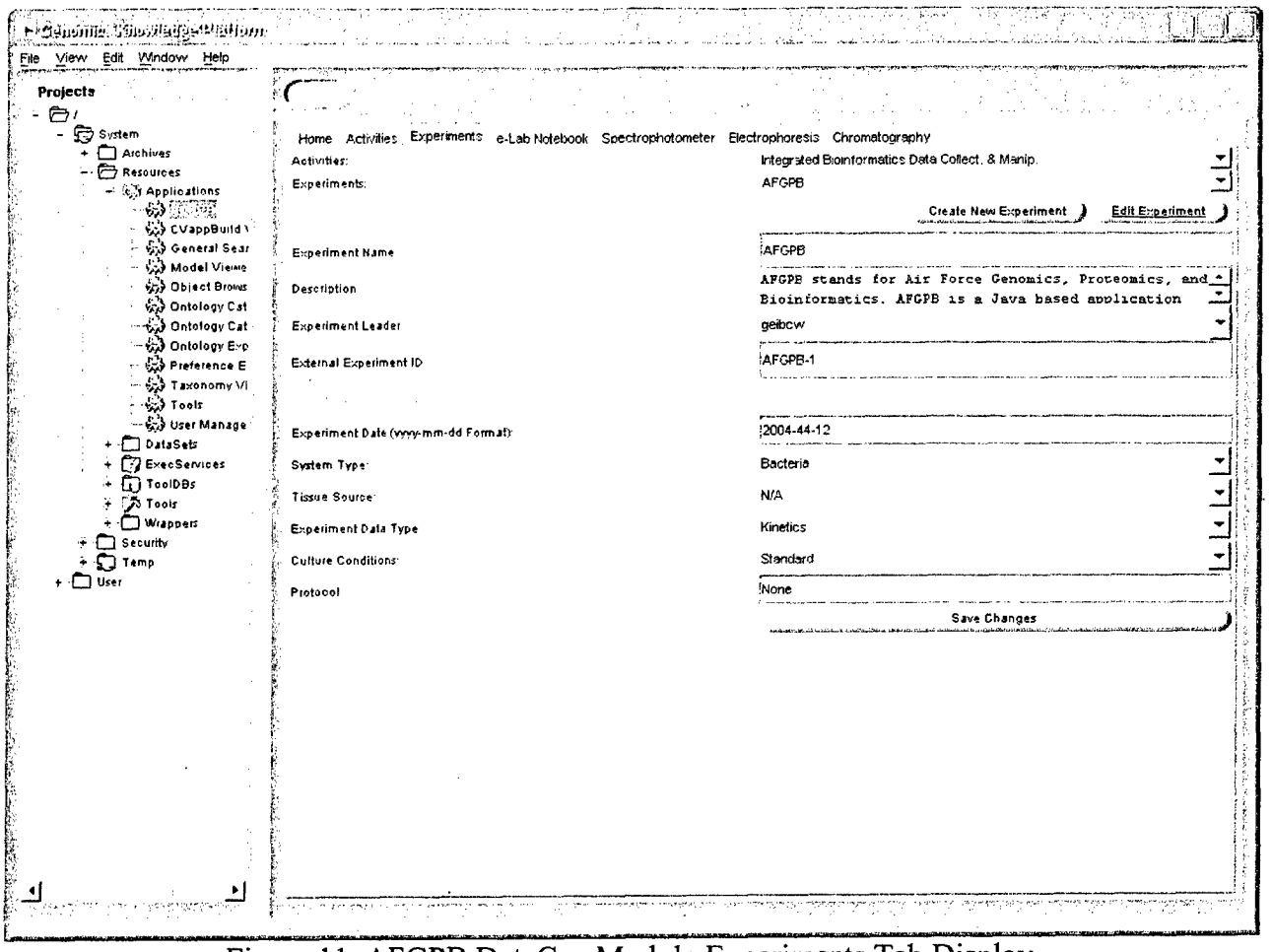

Figure 11. AFGPB DataCap Module Experiments Tab Display

The Experiments tab supports the unique requirements of the Applied Biotechnology Branch to capture experimental data within the Acero GKP. It represents a significant departure from the original P&G GPB application and necessitates an extension of the BOM. The program flow of the Experiments tab is found in Figure 12.

AFGPB Program Flow - Experiment Panel

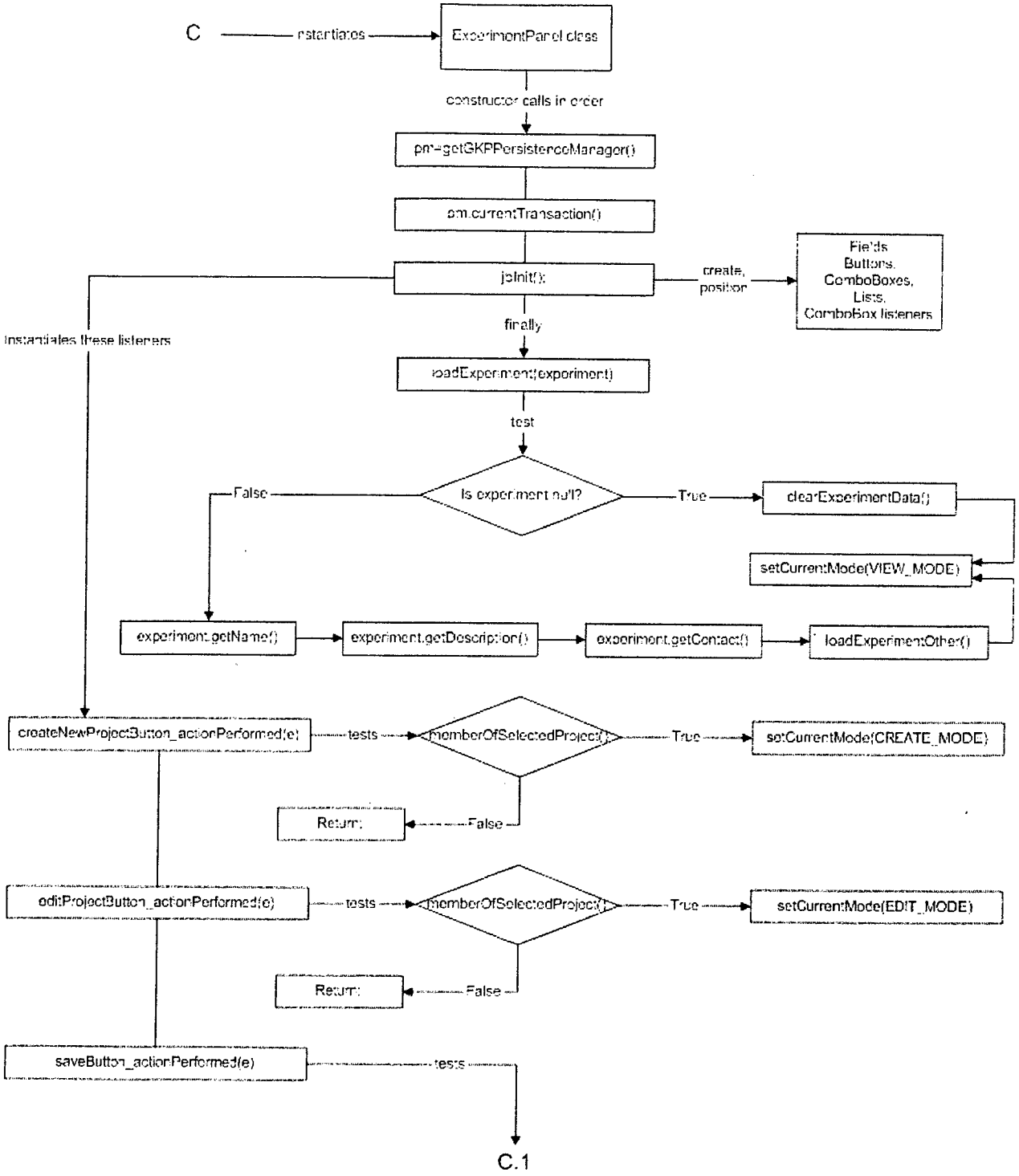

Figure 12. AFGPB DataCap Module Experiment Panel Class Program Flow

When the tabs_stateChanged listener in afgpb.class detects that a user has selected the Experiments tab, the listener initiates a call to a new instance of ExperimentPanel.class. As there are a number of methods that are making Java Data Object (JDO) calls to the database via JDO methods, the constructor has been configured to create a Persistence Manager object via a call to the getGKPPersistenceManager method, and then create a Transaction object via a call from the Persistence Manager object to Transaction Manager. These objects are used extensively to add, update, or remove information from the Oracle database. The constructor then calls the jbInit method to establish the GUI interface with fields, buttons, combo boxes, lists, and combo box listeners. The method while laying out the GUI objects on the screen, also gets information like the list of Activities (Projects). It then populates the ProjectComboBox list, as well as getting the list of Experiments, for the first Activity that displays by default, and populating the experimentComboBox. The jbInit method then instantiates a series of action listeners for buttons. It also instantiates listeners for the Project (Activity) combo box and the Experiment combo box. Finally, the jbInit method makes a call to loadExperiment, passing in the currently selected (default display) experiment. If the variable is null (no experiment exists), the method calls the clearExperimentData method which clears all the data entry fields and then runs the setCurrentMode method passing in the constant VIEW_MODE. If however, the experiment is not null (the most common case), then the method goes about gathering the name, description, the contact and populating the appropriate fields. A call is made to loadExperimentOther method and a JDO query is sent via the Transaction Manager to recover experiment date, cell type, source, experiment data type, culture conditions and protocol, and populating the appropriate fields. This data is stored in the new extended table called Experiment. Upon return of the loadExperimentOther method, the setCurrentMode method is called with the constant VIEW_MODE, being passed.

When the user clicks the Create New Experiment button in the DDK, the createNewProjectButton listener is activated. The listener in turn calls the memberOfSelectedProject method and tests to see if the user is a member of the currently selected activity. If the user is a member, the method returns a Boolean true to the listener and then the listener calls the setCurrentMode method and passes in the constant CREATE_MODE. If the member is not a member of the current activity, the memberOfSelectedProject method displays a dialog stating the user is not a member and then upon acknowledgement, returns a Boolean false to the listener which then exits without changing the setting of setCurrentMode.

Clicking the Edit Experiment button in the DDK triggers the editProjectButton listener. The method called by the listener is functionally identical to the methods called by the createNewProjectButton listener. However, the only difference between the methods is that in the Edit instance the setCurrentMode method is passed the EDIT_MODE constant. In Edit mode, text from the selected experiment goes from being grayed out to active and editable. The setCurrentMode method when EDIT_MODE is passed in, does not call the clearExperimentData method.

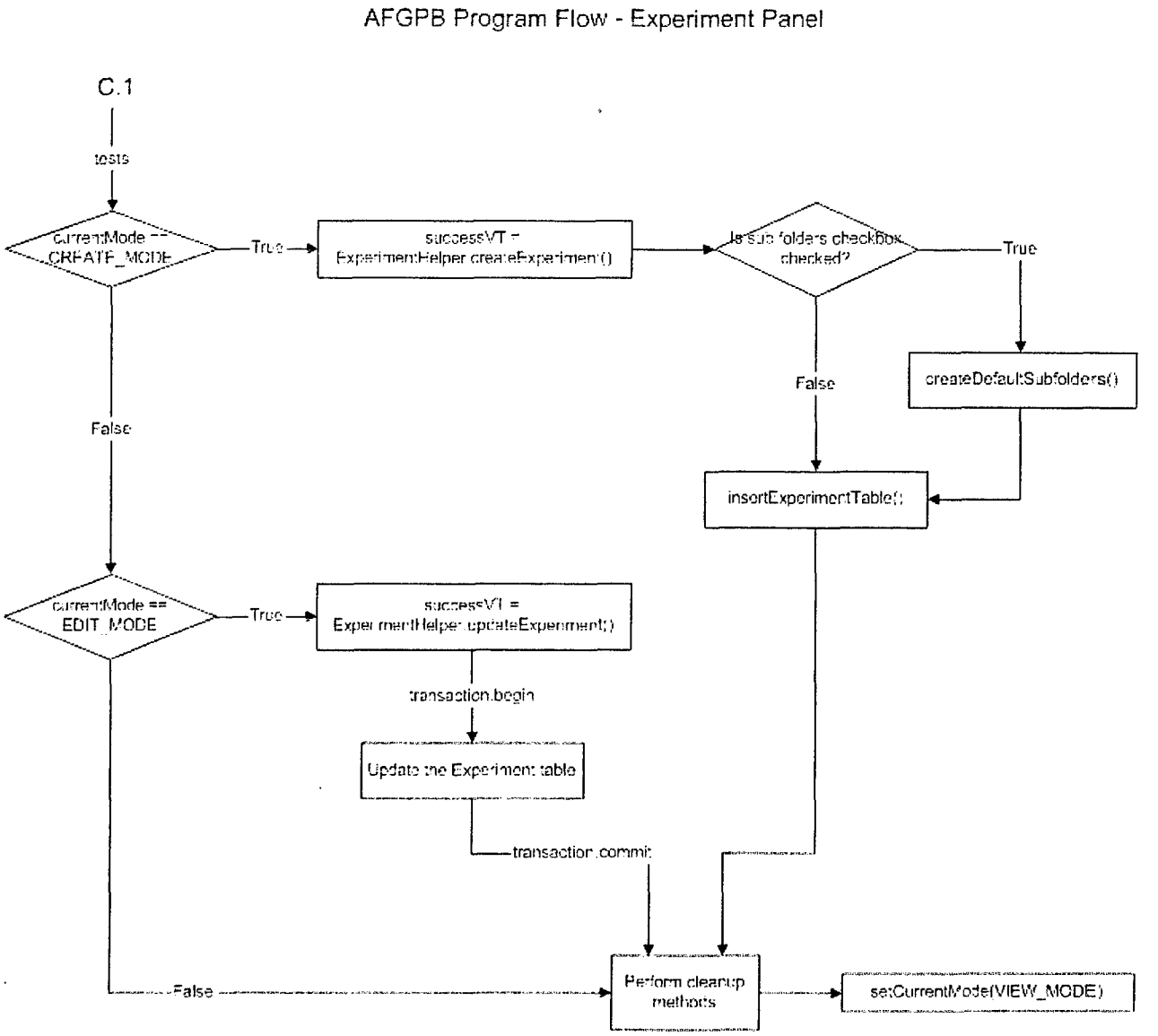

Figure 13. AFGPB DataCap Module Experiment Panel Class Program Flow Continued

The Save Changes button on the Experiments DDK page activates the saveButton listener. The program flow is as shown above in Figure 13. The listener method conducts tests to determine if the current mode of the application is in CREATE_MODE, and if false, then it tests to see if in EDIT_MODE. If the mode is CREATE_MODE, then the method makes a call to the ExperimentHelper class and its createExperiment method. This method sets up properties and then creates the container within the Workbench. Upon return of the method, the listener method determines if the create default subfolders checkbox is checked. If true, the listener calls the ExperimentHelper class method createDefaultSubfolders. This method then creates the containers (folders) listed below

the named Experiment container in the Workbench. This then moves on to call the insertExperimentTable method. If the checkbox is not set, then the method calls the insertExperimentTable method. The insertExperimentTable method takes the entries in the various combo boxes and list boxes and via the persistence manager and transaction manager, passes data via the JDO Experiment class which in turn takes the data and loads it into the Experiment table in the Oracle database. Upon return from the insertExperimentTable method, the listener performs a number of cleanup methods to refresh the view of the Workbench with any changes, and then calls setCurrentMode passing in the constant VIEW_MODE. The listener then exits at this point.

When the listener tests for the current mode, and the test for CREATE_MODE returns false, then the listener tests for EDIT_MODE. If this test also returns false, the listener method then simply goes through the process of cleanup and refreshes the appearance of the Workbench as in CREATE_MODE case. However, if the test for EDIT_MODE returns true, the listener method then calls the ExperimentHelper class method, updateExperiment. This method updates properties much the same as the createExperiment method, however, this method does not create new subfolders. Once the updateExperiment method returns, the listener then starts a transaction, and makes updates to the Experiment table via calls to the JDO based Experiment class. When the calls are complete, the transaction is committed (or rolled back if errors), and the listener method then performs clean up and refreshes the view of the DDK Workbench. The listener methods final act is to set the current mode to VIEW_MODE.

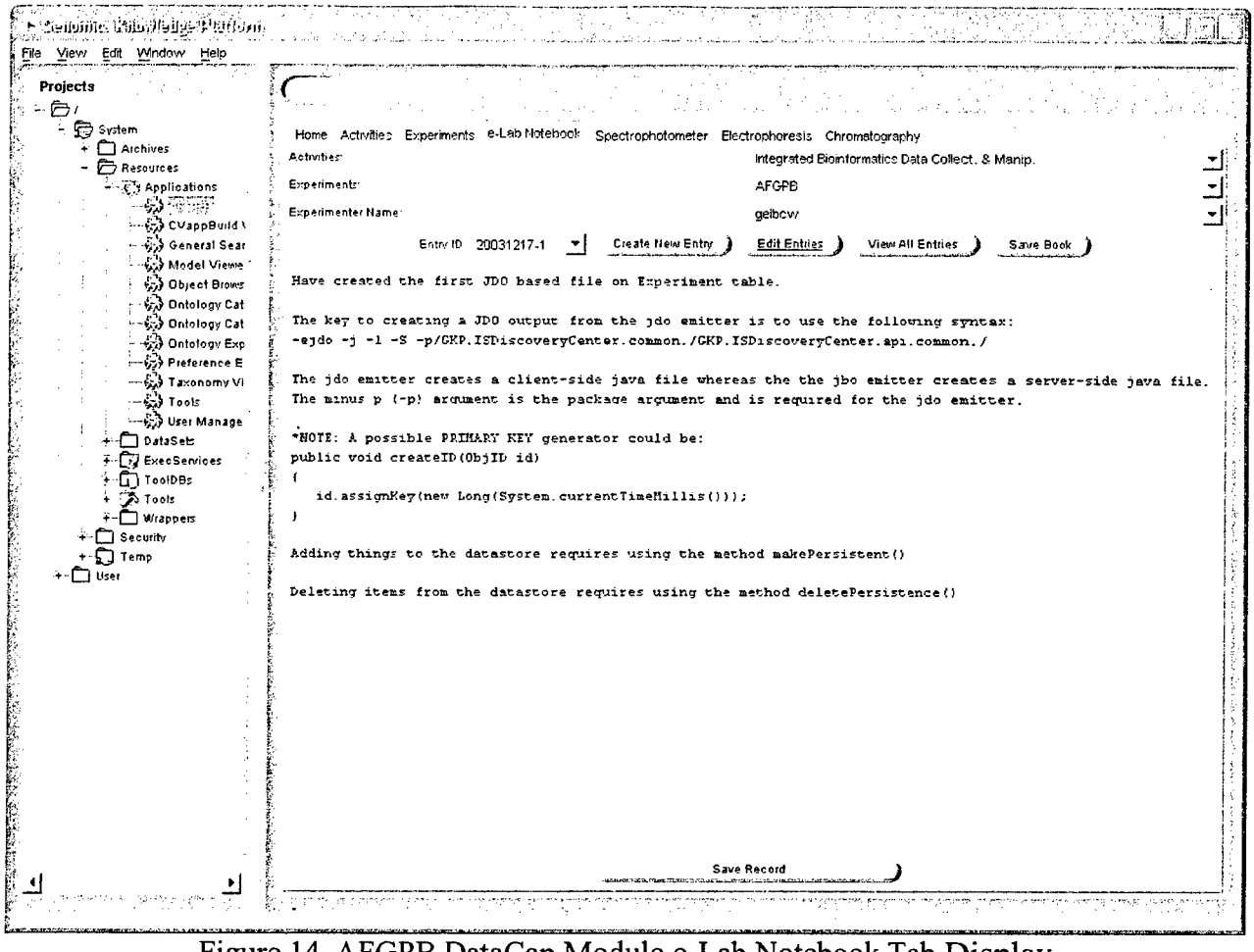

Figure 14. AFGPB DataCap Module e-Lab Notebook Tab Display

3.6.3 e-Lab Notebook Tab

Figure 14 shows the appearance of the e-Lab Notebook tab. The original Procter & Gamble GPB application does not have anyway to capture laboratory notebook data, thus the e-Lab Notebook tab was created. The e-Lab Notebook provides a means for researchers to capture the text notes and comments that are normally written in an individual paper lab notebook. In the Phase II effort, a linkage will be made to graphics stored in the containers within the Workbench. For each notebook entry, the text area for typing notes is limited to 4000 characters. This is because the contents of the text area are captured and stored in an Annotation table entry. The e-Lab Notebook offers additional features, e.g., being able to review all entries from all notebooks with respect to a given experiment. Additionally, all entries for a given experiment can be saved off line as a MS Word file for review and development of reports and papers.

By selecting the e-Lab Notebook tab in the DDK, the user activates the instantiation of the eLabBookPanel class, and the program flow as outlined in Figure 15. As in the ExperimentPanel class previously discussed, the eLabBookPanel class constructor creates a PersistenceManager object, as well as a currentTransaction object. The constructor then

calls the jbInit method which builds the GUI screen within the DDK. The jbInit method creates and places fields, buttons, Combo Boxes, lists and a number of listeners. Three listeners detect changes made in the selected item in the three combo boxes, while the rest are to detect the action of button clicks to the various buttons located on the tab page. Once the GUI portion has been established by the jbInit method, the method calls the loadExperiment method. The loadExperiment method tests to see if no experiment is chosen (null) and if true, then runs the clearExperimentData method to ensure all fields are blank, and then upon return, the method calls setCurrentMode, passing in the VIEW_MODE constant. If however the test to see if experiment is null (that is there is an experiment selected), the method gets the primary activity contact name, then loads the experimentLeaderComboBox with the list of approved persons for that experiment, and then calls getExperimentID which loads a global variable with the OID for the given experiment as stored within the Experiment table in the database. Once these events have completed, the loadExperiment method calls setCurrentMode, passing in the VIEW_MODE constant.

When the user clicks the Create New Entry button, the createNewProjectButton listener is activated. The listener method first determines whether the user is a member of the currently selected activity through a call to the memberOfSelectedProject method. If the user is not a member, the method displays an error message stating the user is not a member and returns a Boolean false to the listener method test. The listener method upon getting a false simply returns. On the other hand, if the user is a member of the currently selected activity, then the memberOfSelectedProject method returns a Boolean true and passes on to the next test, which is to determine if the user is an Admin user. The Boolean state is determined in the first steps of the listener method by a call to the GKP security system to determine if the user is an admin. If the user is not an admin, the listener method then tests to see if the user listed and the user making the request are the same. If they are not, then the listener method generates an error message and then returns. If the listed user is the same as the requesting user, then the method does a setCurrentMode passing in the CREATE_MODE constant. Likewise, if the user is an admin user, the listener method does a setCurrentMode and passes the CREATE_MODE constant.

The editProjectButton listener method is linked to the Edit Entries button. When clicked, the listener method performs the exact same sequence of calls to the same methods as the createNewProjectButton listener method. However, instead of passing the CREATE_MODE constant to the setCurrentMode method, this listener passes the EDIT_MODE constant instead.

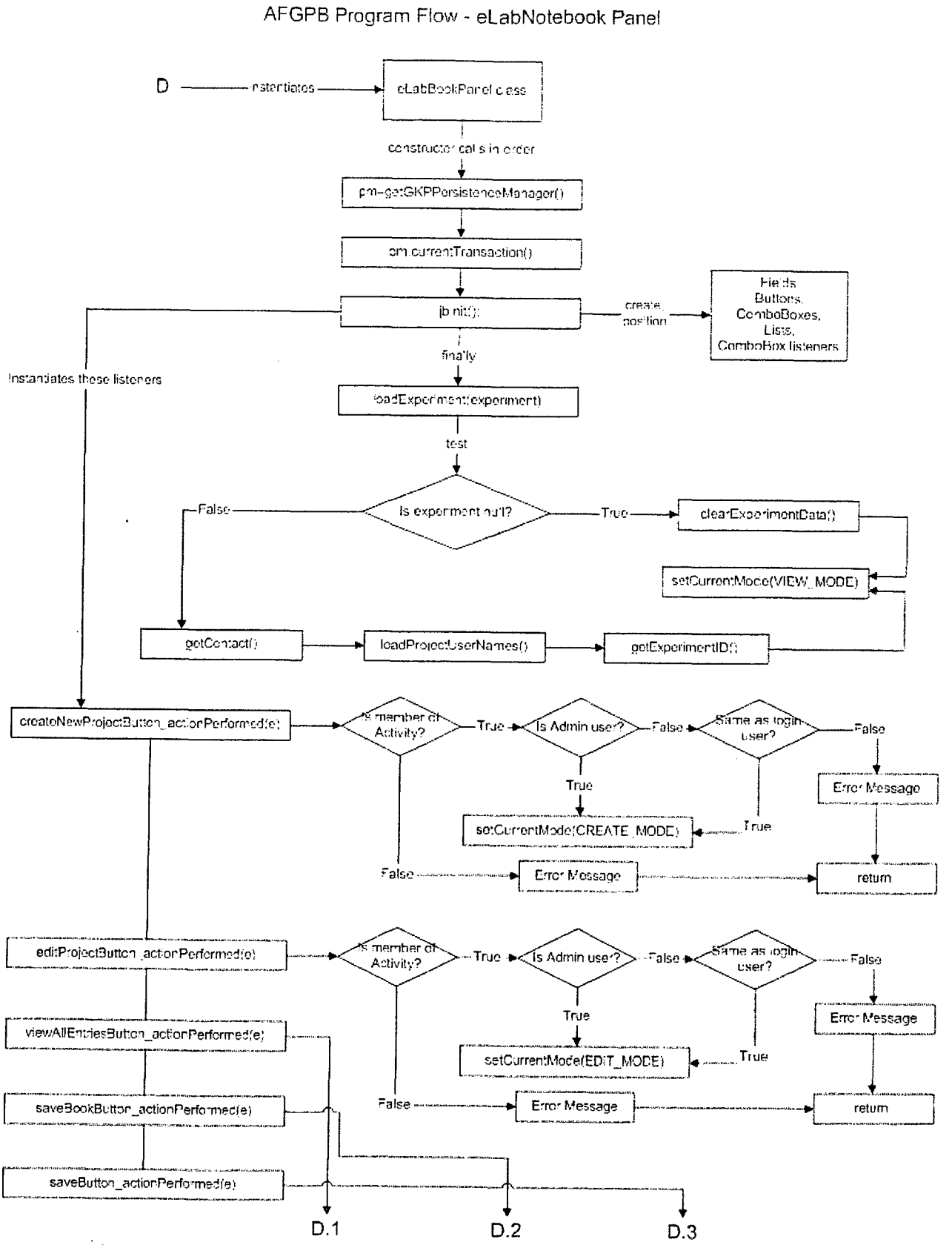

Figure 15a. AFGPB DataCap Module e-Lab Book Panel Class Program Flow

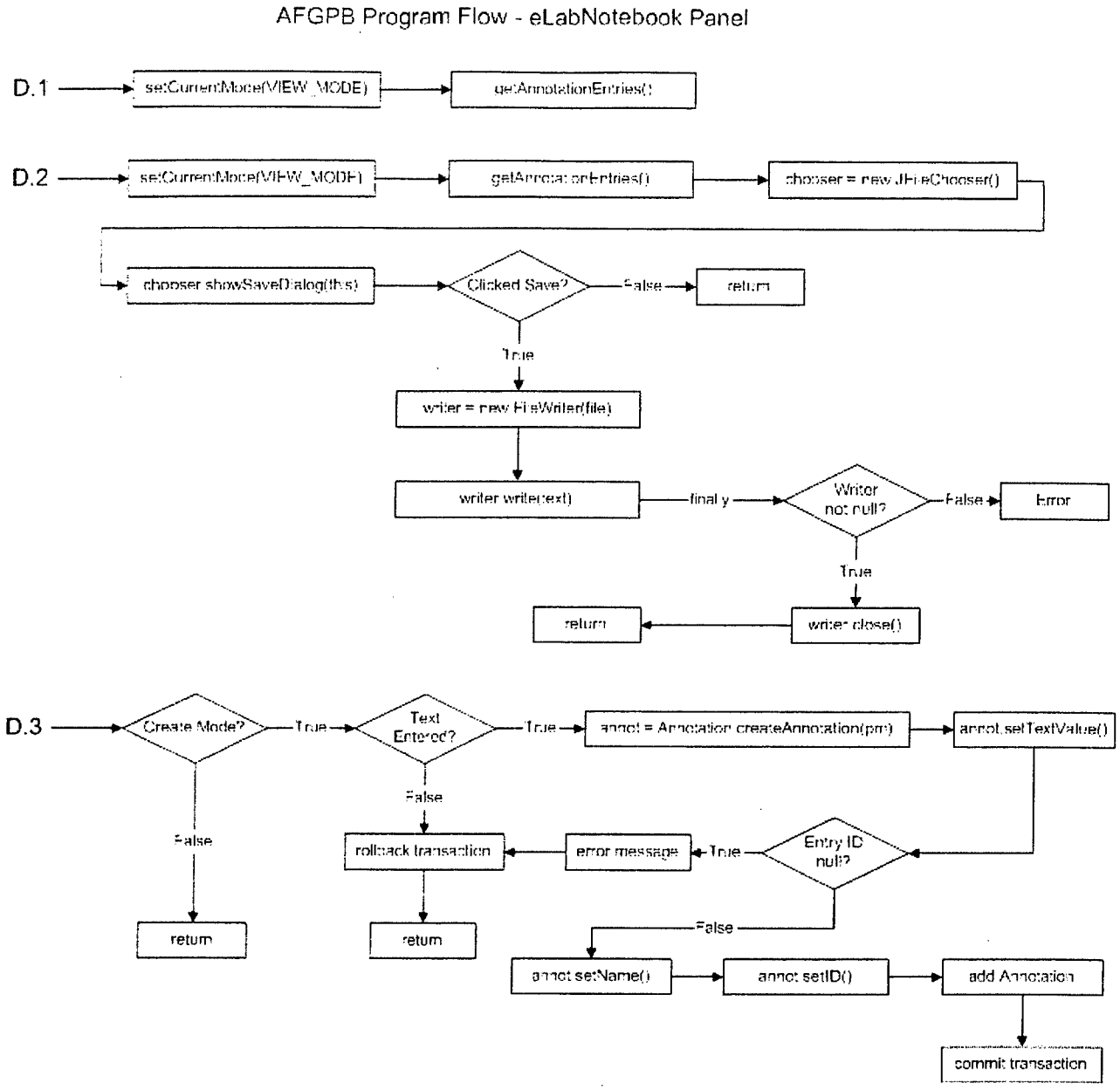

Figure 15b. AFGPB DataCap Module e-Lab Book Panel Class Program Flow Continued

Clicking the View All Entries button activates the viewAllEntriesButton listener method. This listener calls the setCurrentMode method and passes the VIEW_MODE constant. The listener then calls the getAnnotationEntries method. The getAnnotationEntries method first calls clearExperimentData, which clears the text area and the Entry ID field. The method then begins a transaction, runs a query to get all of the entries for the selected experiment for all users that have made an entry, and finally sorts and formats the entries and fills the results into the text area. Once the data has been copied into the text area, the transaction is rolled back to release memory to the system.

The saveBookButton listener starts the process to capture all the lab notebook text currently stored in the database for a given experiment, and format it and save it as a file in Microsoft Word format. This listener calls the setCurrentMode method and passes the VIEW_MODE constant. As in the viewAllEntries listener, the saveBookButton listener calls the getAnnotationEntries method, which completes all of the actions as described in the previous paragraph. Upon return from the getAnnotationEntries method call, a new instance of JFileChooser is created. JFileChooser creates the familiar File Save dialog box common in Windows. Once the user enters a file name and clicks the Save button, a new Java FileWriter instance is created which creates the file at the designated location and appends the information into it. A check is made to ensure the writer is not null, and if true, closes the writer and releases the system memory, then returns from the method. In the event the writer process fails and the writer is null, an error message is returned.

The SaveButton listener is started when the user clicks the Save button. The Save button is grayed out and not useable unless the user has set the current mode to CREATE_MODE, by clicking the New Entry button. If the current mode is CREATE_MODE, the listener determines if any text has been entered in the Notebook workspace. If not, the listener executes a rollback action and then returns. If text is entered, the listener instantiates an instance of Annotation through a createAnnotation method call. The text from the workspace is captured and added to the Annotation instance. The listener then checks for a valid entry ID. Entry IDs are created automatically when the user clicks the New Entry button. If the Entry ID is null, the listener fires an error message, rollbacks the transaction, and returns. With an entry in the Entry ID, the listener adds additional information such as the name, the ID number, and commits the transaction, saving the notebook entry into the Annotation table of the database.

This concludes the Discussion section on the AFGPB DataCap Phase I module program flow. This section covered all the relevant information on the inner workings of the module.

4.0 CONCLUSIONS

The Air Force Genomics, Proteomics, and Bioinformatics System(AFGPB) DataCap Phase I module provides the Applied Biotechnology Branch with a state of the art data collection module. The Phase I module allows the creation of Activities and Experiments, as well as the ability to capture the disparate data associated with Experiments from the various paper lab notebooks in use in the laboratory. The future Phase II effort will provide the capability to capture data created by the various lab instruments used.

5.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFGPB	Air Force Genomics, Proteomics, Bioinformatics System
AFRL	Air Force Research Laboratory
API	Application Programming Interface
BOM	The Acero Biological Object Model
CD&E	Cellular Dynamics and Engineering
COTS	Commercial off the Shelf
DDK	The Acero Desktop Development Kit
GHz	Gigaherz
GKP	The Acero Genomics Knowledge Platform
GKPFS	Genomics Knowledge Platform File System
GPB	Genomics, Proteomics, Bioinformatics
GRIP	Genomics Research Infrastructure Partnership
GUI	Graphical User Interface
JBO	Java Bean Object
JDO	Java Data Objects
MB	Megabyte
MOF	Metadata Object File
ODLC	The Acero Object Data Language Compiler
OID	Object ID Number
RAID	Redundant Array of Independent Disks
SQL	Structured Query Language
UML	Uniform Modeling Language

VARCHAR2	Variable Character Type 2 - An Oracle Data Type